EUCLIDEAN DISTANCE INSTRUCTIONS

BACKGROUND OF THE INVENTION

Field of the Invention:

The present invention relates to systems and methods for instruction processing and, more particularly, to systems and methods for providing multiplication computation instruction processing, pursuant to which the difference of two values transferred directly from a memory location in memory to an execution unit are squared.

Description of Prior Art:

Processors, including microprocessors, digital signal processors and microcontrollers, operate by running software programs that are embodied in one or more series of instructions stored in a memory. The processors run the software by fetching instructions from the series of instructions, decoding the instructions and executing the instructions. Processors, including digital signal processors, are conventionally adept at processing instructions that perform mathematical computations on integers specified as a data word. For example, some processors are adept at performing simple mathematical computation, such as sum and difference operations, while others are adept at performing complex computations, such as multiplicative operations. In general, mathematical operations are performed by mathematical execution units in accordance with a set of mathematical instructions. Various mathematical instructions in the set of mathematical instruction are responsible for the performance of various aspects of the mathematical operation. The mathematical operation instructions specify the memory locations of a set

of source operands contained in memory upon which to perform the mathematical operation. Typically, the set of source operands is transferred from the memory locations to registers prior to execution of the mathematical operation by appropriate mathematical execution units. These mathematical operations make inefficient use of processor resources and tend to reduce the performance of the processor. The inefficiency is due to the number of instructions required to perform a mathematical operation as well as the transference of the set of source operands from the memory locations in memory to registers inorder to perform the mathematical operation.

One such mathematical operation is an Euclidean distance operation, pursuant to which the difference of two operands is squared and the difference of two operands is calculated. Conventionally, this instruction would have to be implemented using a series of mathematical instructions. For example, a separate instruction would be required to calculate the difference of two operands, square the difference of the two operands, and store the square of the difference of the two operands. In addition, the two operands would be transferred from each of their respective memory locations in memory to registers prior to execution of the mathematical operations instructions by appropriate mathematical execution units. This technique requires multiple processor cycles and mathematical instructions, and accordingly, is inefficient.

There is a need for a new method of implementing mathematical operations within a processor that makes efficient use of processor cycles. There is a further need for a new method of implementing mathematical operations using a reduced number of mathematical instructions. There is also a need for a new method of implementing mathematical

operations that transfer operands directly from data memory to mathematical execution units. There is a need for a new method of implementing mathematical operations that concurrently employs processor MCU ALU execution unit and DSP multiplier execution unit resources.

SUMMARY OF THE INVENTION

According to embodiments of the present invention, a method and a processor for processing mathematical operation instructions are provided. There are two mathematical operation instructions, each for fetching source operands directly from data memory and registers for immediate execution by appropriate mathematical execution units. In addition, each mathematical operation instruction performs a difference and multiplication operation simultaneously concurrently employing mathematical execution units. The instructions may specify the location of the source operands as well as a destination location for results of mathematical operations. These instructions may be executed in one processor cycle employing mathematical operation logic provided within the processor.

According to an embodiment of the present invention, a method of processing a multiplication operation instruction includes fetching and decoding a multiplication operation instruction. The method further includes executing the multiplication operation instruction on a multiplication source operand to generate a square of the multiplication source operand. A multiplication source operand register specified in the multiplication operation instruction contains the multiplication operand. The method further includes storing a result in a target accumulator specified in the multiplication operation instruction.

The multiplication source operand is transferred directly from the multiplication operand source register to a multiplication execution unit for immediate generation of the square of the multiplication operand. The method further includes setting an accumulator status flag during the execution of the multiplication operation instruction.

According to another embodiment of the present invention, a method of processing a multiplication operation instruction includes generating a difference output between a minuend source operand and a subtrahend source operand and storing the difference output in a difference register specified in the multiplication operation instruction. The minuend operand and the subtrahend operand are fetched during execution of the multiplication operation instruction. The multiplication operation instruction specifies a first source register containing an address for the minuend operand in data memory and a second source register containing the address of the subtrahend operand in data memory. The minuend operand and subtrahend operand may be transferred directly from the address locations in memory to an arithmetic execution unit for immediate generation of the difference. The multiplication operation instruction may be an Euclidean Distance operation, where the result of the operation is the square of the multiplication source operand. Alternatively, the multiplication operation instruction may be an Euclidean Distance Accumulate operation, where the result of the operation is the sum of the square of the multiplication source operand and an addition operand.

According to another embodiment of the present invention, a method of processing a multiplication operation instruction includes modifying the address contained in the first source register to contain an address for a next minuend operand for computing a

difference during execution of a subsequent multiplication operation instruction. The method further includes modifying the address contained in the second source register to contain an address for a next subtrahend operand for computing the difference during execution of the subsequent multiplication operation instruction.

According to an embodiment of the present invention, a processor for performing multiplication operation instructions includes a program memory for storing a multiplication operation instruction, a program counter for identifying current instructions for processing, and a Digital Signal Processing unit (DSP) and ALU for executing the multiplication operation instructions within the program memory. The DSP unit includes DSP logic for executing the multiplication operation instruction on a multiplication source operand to generate a square of the multiplication source operand. The multiplication source operand is contained in a multiplication source operand register specified in the multiplication operation instruction. A target accumulator stores a result output. The target accumulator is specified in the multiplication operation instruction. The multiplication source operand is transferred directly from the multiplication source operand register to a multiplication execution unit for immediate generation of the square of the multiplication source operand.

BRIEF DESCRIPTION OF THE DRAWINGS

The above described features and advantages of the present invention will be more fully appreciated with reference to the detailed description and appended figures in which:

Fig. 1 depicts a functional block diagram of an embodiment of a processor chip within which embodiments of the present invention may find application;

Fig. 2 depicts a functional block diagram of a data busing scheme for use in a processor, which has a microcontroller and a digital signal processing engine, within which embodiments of the present invention may find application;

Fig. 3 depicts a functional block diagram of a processor configuration for processing mathematical operation instructions according to embodiments of the present invention;

Fig. 4 depicts a method of processing mathematical operation instructions according to embodiments of the present invention; and

Fig. 5 depicts a table of mathematical operation instructions according to embodiments of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

According to embodiments of the present invention, a method and a processor for processing mathematical operation instructions are provided. There are two mathematical operation instructions, each for fetching source operands directly from data memory and registers for immediate execution by appropriate mathematical execution units. In addition, each mathematical operation instruction performs a difference and multiplication operation simultaneously, concurrently employing mathematical execution units. In addition, the instructions may specify the location of the source operands as well as destination locations for results. These instructions may be executed in one processor

cycle employing mathematical operation logic provided within the processor. These instructions may improve performance over conventional techniques by several times.

In order to describe embodiments of mathematical operation instruction processing, an overview of pertinent processor elements is first presented with reference to Figs. 1 and 2. The mathematical operation instructions and instruction processing are then described more particularly with reference to Figs. 3-5.

Overview of Processor Elements

Fig. 1 depicts a functional block diagram of an embodiment of a processor chip within which the present invention may find application. Referring to Fig. 1, a processor 100 is coupled to external devices/systems 140. The processor 100 may be any type of processor including, for example, a digital signal processor (DSP), a microprocessor, a microcontroller or combinations thereof. The external devices 140 may be any type of systems or devices including input/output devices such as keyboards, displays, speakers, microphones, memory, or other systems which may or may not include processors. Moreover, the processor 100 and the external devices 140 may together comprise a stand alone system.

The processor 100 includes a program memory 105, an instruction fetch/decode unit 110, instruction execution units 115, data memory and registers 120, peripherals 125, data I/O 130, and a program counter and loop control unit 135. The bus 150, which may include one or more common buses, communicates data between the units as shown.

The program memory 105 stores software embodied in program instructions for execution by the processor 100. The program memory 105 may comprise any type of nonvolatile memory such as a read only memory (ROM), a programmable read only memory (PROM), an electrically programmable or an electrically programmable and erasable read only memory (EPROM or EEPROM) or flash memory. In addition, the program memory 105 may be supplemented with external nonvolatile memory 145 as shown to increase the complexity of software available to the processor 100. Alternatively, the program memory may be volatile memory which receives program instructions from, for example, an external non-volatile memory 145. When the program memory 105 is nonvolatile memory, the program memory may be programmed at the time of manufacturing the processor 100 or prior to or during implementation of the processor 100 within a system. In the latter scenario, the processor 100 may be programmed through a process called in-line serial programming.

The instruction fetch/decode unit 110 is coupled to the program memory 105, the instruction execution units 115 and the data memory 120. Coupled to the program memory 105 and the bus 150 is the program counter and loop control unit 135. The instruction fetch/decode unit 110 fetches the instructions from the program memory 105 specified by the address value contained in the program counter 135. The instruction fetch/decode unit 110 then decodes the fetched instructions and sends the decoded instructions to the appropriate execution unit 115. The instruction fetch/decode unit 110 may also send operand information including addresses of data to the data memory 120 and to functional elements that access the registers.

The program counter and loop control unit 135 includes a program counter register (not shown) which stores an address of the next instruction to be fetched. During normal instruction processing, the program counter register may be incremented to cause sequential instructions to be fetched. Alternatively, the program counter value may be altered by loading a new value into it via the bus 150. The new value may be derived based on decoding and executing a flow control instruction such as, for example, a branch instruction. In addition, the loop control portion of the program counter and loop control unit 135 may be used to provide repeat instruction processing and repeat loop control as further described below.

The instruction execution units 115 receive the decoded mathematical instructions from the instruction fetch/decode unit 110 and thereafter execute the decoded mathematical instructions. As part of this process, the execution units may retrieve a set of source operands via the bus 150 from data memory and registers 120. During instruction processing, such as mathematical operation instructions, the set of source operands may be fetched from data memory and registers 120 as specified in the mathematical operation instructions. The set of set of source operands may be transferred from the data memory and registers 120 to registers prior to delivery to the execution units for processing. Alternatively, the set of source operands may be delivered directly from the data memory and registers 120 to the execution units for processing. Execution units may also produce outputs to registers and/or the data memory 120. The execution units may include an arithmetic logic unit (ALU) such as those typically found in a microcontroller. The execution units may also include a digital signal processing engine including a multiply

and accumulate unit (MAC), a floating point processor, an integer processor or any other convenient execution unit. A preferred embodiment of the execution units and their interaction with the bus 150, which may include one or more buses, is presented in more detail below with reference to Fig. 2.

The data memory and registers 120 are volatile memory and are used to store data used and generated by the execution units. The data memory 120 and program memory 105 are preferably separate memories for storing data and program instructions respectively. This format is a known generally as a Harvard architecture. It is noted, however, that according to the present invention, the architecture may be a Von-Neuman architecture or a modified Harvard architecture which permits the use of some program space for data space. A dotted line is shown, for example, connecting the program memory 105 to the bus 150. This path may include logic for aligning data reads from program space such as, for example, during table reads from program space to data memory 120.

Referring again to Fig. 1, a plurality of peripherals 125 on the processor may be coupled to the bus 125. The peripherals may include, for example, analog to digital converters, timers, bus interfaces and protocols such as, for example, the controller area network (CAN) protocol or the Universal Serial Bus (USB) protocol and other peripherals. The peripherals exchange data over the bus 150 with the other units.

The data I/O unit 130 may include transceivers and other logic for interfacing with the external devices/systems 140. The data I/O unit 130 may further include functionality

to permit in circuit serial programming of the Program memory through the data I/O unit 130.

Fig. 2 depicts a functional block diagram of a data busing scheme for use in a processor 100, such as that shown in Fig. 1, which has an integrated microcontroller arithmetic logic unit (ALU) 270 and a digital signal processing (DSP) engine 230. This configuration may be used to integrate DSP functionality to an existing microcontroller core. Referring to Fig. 2, the data memory 120 of Fig. 1 is implemented as two separate memories: an X-memory 210 and a Y-memory 220, each being respectively addressable by an X-address generator 250 and a Y-address generator 260. The X-address generator may also permit addressing the Y-memory space thus making the data space appear like a single contiguous memory space when addressed from the X address generator. The bus 150 may be implemented as two buses, one for each of the X and Y memory, to permit simultaneous fetching of data from the X and Y memories.

The W registers 240 are general purpose address and/or data registers. The DSP engine 230 is coupled to both the X and Y memory buses and to the W registers 240. The DSP engine 230 may simultaneously fetch data from each the X and Y memory, execute instructions which operate on the simultaneously fetched data and write the result to an accumulator (not shown) and write a prior result to X or Y memory or to the W registers 240 within a single processor cycle.

In one embodiment, the ALU 270 may be coupled only to the X memory bus and may only fetch data from the X bus. However, the X and Y memories 210 and 220 may be addressed as a single memory space by the X address generator in order to make the data

memory segregation transparent to the ALU 270. The memory locations within the X and Y memories may be addressed by values stored in the W registers 240.

Any processor clocking scheme may be implemented for fetching and executing instructions. A specific example follows, however, to illustrate an embodiment of the present invention. Each instruction cycle is comprised of four Q clock cycles Q1 – Q4. The four phase Q cycles provide timing signals to coordinate the decode, read, process data and write data portions of each instruction cycle.

According to one embodiment of the processor 100, the processor 100 concurrently performs two operations – it fetches the next instruction and executes the present instruction. Accordingly, the two processes occur simultaneously. The following sequence of events may comprise, for example, the fetch instruction cycle:

Q1: Fetch Instruction
Q2: Fetch Instruction
Q3: Fetch Instruction
Q4: Latch Instruction into prefetch register, Increment PC

The following sequence of events may comprise, for example, the execute instruction cycle for a single operand instruction:

Q1: latch instruction into IR, decode and determine addresses of operand data
Q2: fetch operand
Q3: execute function specified by instruction and calculate destination address for data
Q4: write result to destination

The following sequence of events may comprise, for example, the execute instruction cycle for a dual operand instruction using a data pre-fetch mechanism. These instructions pre-fetch the dual operands simultaneously from the X and Y data memories

and store them into registers specified in the instruction. They simultaneously allow instruction execution on the operands fetched during the previous cycle.

Q1: latch instruction into IR, decode and determine addresses of operand data
Q2: pre-fetch operands into specified registers, execute operation in instruction
Q3: execute operation in instruction, calculate destination address for data
Q4: complete execution, write result to destination

Mathematical Operation Instruction Processing

Fig. 3 depicts a functional block diagram of a processor for processing a mathematical operation instruction according to an embodiment of the present invention. Referring to Fig. 3, the processor includes a program memory 300 for storing instructions such as the mathematical operation instructions depicted in Fig. 5. The processor also includes a program counter 305 which stores a pointer to the next program instruction that is to be fetched. The processor further includes an instruction register 315 for storing an instruction for execution that has been fetched from the program memory 300. The processor also includes an instruction decoder 320, an arithmetic logic unit (ALU) 325a, a DSP unit 325b, registers 345, a status register 350 and accumulators 360.

Registers 345 contain data that the processor generates and employs during processing mathematical operation instructions. The registers 345 may include a set of 16-bit W registers defined for mathematical operation instructions. The set of W registers may contain data including an operand and an effective address of an operand in data memory 355. The effective address of an operand in memory can be specified as an address or an address plus an offset. The effective addresses contained in W registers may be modified

by a constant value following the execution of a mathematical operation instruction by execution units. Operands contained in W registers may be transferred directly to execution units, such as the DSP unit 325b and ALU unit 325a, for processing in accordance with a mathematical operation instruction. Processing of operands by DSP unit 325b and ALU unit 325a is performed concurrently. The direct transfer of operands and concurrent processing of operands by DSP unit 325b and ALU unit 325b saves processor cycles and resources.

Accumulators 360 contain data generated and produced by the DSP unit 325b as output during execution of a mathematical operation instruction. For example, each of the accumulators can contain the output of an executed square mathematical operation instruction. The Accumulators may be two 40-bit accumulators, one of which is specified by bits in the mathematical operation instruction as a target accumulator for output produced by the DSP unit 325b.

Status register 350 contains status bits that indicate the status of processor elements and operations. Status register 350 may be a 16-bit status register. The status register 350 may be separated into a lower segment and an upper segment. The processor operations for which status may be indicated include MCU ALU, DSP Adder/Subtractor, repeat, and Do loop. The processor elements for which status may be indicated include accumulators. A set of bits in the status register 350 can indicate the overflow and saturation status of accumulators due to DSP Adder/Subtractor operations.

The overflow statuses indicated include accumulator A overflowed into guard bits, and accumulator B overflowed into guard bits. The overflow status bits are modified each

time data interacts with the DSP Adder/Subtractor. When the overflow status bits are set, such as with a value of 1, they indicate the most recent mathematical operation has overflowed into guard bits of an accumulator. The saturation statuses indicated include accumulator A saturated, and accumulator B saturated. The saturation status bits are modified each time data interacts with the DSP Adder/Subtractor and may be cleared by a programmer. When the saturation status bits are set, such as with a value of 1, they indicate that the accumulator has overflowed its maximum range.

The instruction decoder 320 decodes mathematical operation instructions that are stored in the instruction register 315. Based on the bits in the mathematical operation instruction, the instruction decoder 320 selectively activates logic within the ALU 325a and DSP 325b for fetching source operands, performing the specified operation on the source operands, and returning the outputs to appropriate memory location. The instruction decoder 320 decodes particular bits in the mathematical instructions and sends control signals to the ALU 325a and DSP 325b which direct the fetching of the correct source operands specified by operand register bits in the instruction, direct the activation of the correct portion of the ALU logic 335a and DSP logic 335b to carry out the mathematical operation specified by mathematical operation bits in the instruction on the correct source operands, and direct the outputs the mathematical operation to be written to the correct destinations specified by the destination bits in the instruction.

The ALU unit 325a and DSP unit 325b can include registers 330a-330b that receive operands from the registers 345 and/or a data memory 355 depending on the addressing mode used in the instruction. For example in one addressing mode, source operands may

be stored in the registers 345 and fed directly to ALU unit 325a and DSP unit 325b for processing. In another addressing mode, the source operands may be stored in the data memory 355. Alternatively, some source operands may be stored in registers 345 while others may be stored in the data memory 355.

The ALU unit 325a includes ALU logic 335a which can receive inputs from the registers 330b, directly from registers 345 or directly from data memory 355, and produces outputs to the registers 345. The ALU logic 335 executes arithmetic and logic operations, specified by control signals based on bits in a mathematical operation instruction decoded by the instruction decoder, on operands fetched from the registers 345 and/or from the data memory 355. In addition, ALU unit 325a produces outputs to the registers 345, specified by control signals based on destination bits in the instruction decoded by the instruction decoder. In general, the ALU unit 325b processes data in byte or word widths.

The DSP unit 325b includes DSP logic 335b, which can receive inputs from the registers 330b and/or directly from registers 345 or data memory 355 and produces outputs to accumulators 360. The DSP logic 335b executes multiplicative and logic operations, specified by control signals based on bits in a mathematical operation instructions decoded by the instruction decoder on operands fetched from the registers 345 and/or from the data memory 355. In addition, the DSP unit 325a produces an output to a target accumulator, specified by control signals based on destination bits in the instruction decoded by the instruction decoder.

ALU logic 335a and DSP logic 335b are logically separated and are activated upon the execution of one of the mathematical operation instructions shown in Fig. 5. In this

regard, when a mathematical operation instruction, such as one of those depicted in Fig. 5, is present in the instruction decoder 320, the instruction decoder generates control signals which cause the ALU unit and DSP unit to fetch the specified source operands from the registers 345 or from the data memory 355 and which cause the DSP logic 335b and ALU logic 335a to operate on the fetched source operands to produce outputs in accordance with the instruction. The outputs depend upon the instruction executed and the source operands. After generating the outputs, the instruction decoder causes the outputs to be written into the correct registers 345, memory locations within the data memory 355, and/or a target accumulator 360.

The logic may include logic for implementing two different mathematical operation instructions such as those depicted in Fig. 5. Each of these instructions performs a square operation and a difference operation on source operands whose location is specified by source register bits in the mathematical operation instruction as indicated in the table of Fig. 5. Each instruction produces outputs to destination locations specified by destination bits in the mathematical instruction. The logic for implementing each instruction is selectively activated by the instruction decoder 320 when that particular instruction is decoded.

Saturation logic 365 determines the saturation and overflow status of accumulators 360. In this regard, during execution of a mathematical operation the saturation logic analyzes a set of bits for the target accumulator to determine if the mathematical operation performed in accordance with the mathematical operation instruction has produced an output that overflowed into guard bits of the accumulator and/or overflowed the maximum

range for the accumulator. Based on the analysis, overflow and saturation status bits may be set to indicate the overflow and saturation status of the target accumulator.

Fig. 4 depicts a method of processing mathematical operation instructions, such as Euclidean Distance operation instructions, and is best understood when viewed in combination with Fig. 3. Referring to Fig. 4, in step 400, the processor fetches a mathematical operation instruction from the program memory 300. Then in step 410, the instruction decoder 320 decodes the mathematical operation instruction. The mathematical operation instruction may specify a register containing a source multiplication source operand for a square operation, a target accumulator to output the square of the square operation, registers containing the addresses of a minuend and subtrahend operands in the memory for a difference operation, a register to output the difference of the difference operation and an accumulator for a writeback operation.

In step 420, the processor causes control signals to be sent to the ALU logic and DSP logic. The control signals sent are based on bits in the mathematical operation instruction. The control signals indicate the locations of source operands in data memory 355 and registers 345, the destination locations for outputs and the mathematical operations to perform on the source operands. In step 430, the ALU unit 325a and DSP unit 325b fetch source operands from registers 345 or the data memory 355, as specified by source register bits in the mathematical operation instruction. In step 440, the processor may execute a subtraction operation according to the decoded mathematical operation instruction using the source operands to obtain a difference output from the ALU unit 325a. In step 450, the ALU unit stores the difference output to a difference register 345 for

use in a multiplication operation according to a subsequently decoded mathematical operation instruction. In step 460, the processor may execute a multiplication operation according to the decoded mathematical operation instruction using the source operands to obtain a square output from the DSP unit 325b. In step 470, the DSP unit accumulates the square output to a target accumulator 360, as specified by destination bits in the mathematical operation instruction. In step 480, address locations of source operands for the subsequent mathematical operation instruction are modified. In step 490, the status register may be set indicating the overflow and saturation status of the target accumulator.

While specific embodiments of the present invention have been illustrated and described, it will be understood by those having ordinary skill in the art that changes may be made to those embodiments without departing from the spirit and scope of the invention.